Crawl

From SangerWiki

Contents

Documentation

Crawl (Chado Restful Access Webservice Layer) is a data mining tool being developed in Pathogen Informatics, for the purpose of query a chado database.

Any questions, contact Giles Velarde (gv1).

Usage

It has currently 2 general modes of usage. Natively in Python, and indirectly via exposed programmatic interfaces.

- direct programmatic access via the Python scripting language
- indirectly via web services, the command line and wrapper libraries
  - using the command line client
  - via webservices (for external use or AJAX applications)
  - wrapping either of the above with the use of wrapper libraries

Natively in Python

The crawl library is versioned in our psu subversion inside genlib/python/crawl. Most users should already have it in the path. Here's an example of programmatic access to the public read only database:

```
from crawl.api.controllers import Organisms
from crawl.api.db import Queries
from ropy.query import ConnectionFactory

connectionFactory = ConnectionFactory("db.genedb.org", "snapshot", "genedb_ro", "")
organisms = Organisms()
organisms.queries = Queries(connectionFactory)
print organisms.list()
```

in this example a list of organisms and their taxonIDs are being returned.

Webservices

Crawl web-services have been developed with a few real use-cases in mind:

- Allowing remote collaborators to programmatically query our databases routinely.
- Supporting AJAX-based web-applications.
- Supporting the possibility of developing query scripts against our database that can be run outside the Sanger systems.

The crawl library is deployed as [web services (http://t81-omixed.internal.sanger.ac.uk:6666/)], and these are programmatically accessible from any language.

Because these are RESTful (http://en.wikipedia.org/wiki/Representational_State_Transfer) services, they are very simple call and test in your browser. All you need to do is make an HTTP request from whatever language you choose to work in. Here are some examples, but they are by no means exhaustive. If you need help getting web services to work with whatever platform you're using, contact gv1.

Examples

Some example requests:

- listing organisms (http://t81-omixed.internal.sanger.ac.uk:6666/organisms/list) with their taxonIDs
- listing genes (http://t81-omixed.internal.sanger.ac.uk:6666/genes/inorganism?taxonID=420245) in an organism, this query requires a taxonID parameter

All queries default to generating XML. If you want to generate JSON, you just add a .json extension at the end of the URI path, but before the parameters.

- listing organisms (http://t81-omixed.internal.sanger.ac.uk:6666/organisms/list.json) with their taxonIDs
- listing genes (http://t81-omixed.internal.sanger.ac.uk:6666/genes/inorganism.json?taxonID=420245) in an organism, this query requires a taxonID parameter

Reflection

The webservices can be inspected to see what queries are available:

- the top level (http://t81-omixed.internal.sanger.ac.uk:6666/) path will list available query sets
- the genes (http://t81-omixed.internal.sanger.ac.uk:6666/genes) path will list queries available in the "genes" query set

All requests can be made using GET or POST, with the exception of JSONP requests, which by their nature are GET only.

Calling the webservices from PERL

A perl module (Ropy::RopyClient) is available in genlib/perl.

Here is an example of how to call it to make get all the gene mrnas in an organism:

```
use strict;
use warnings;
use Log::Log4perl qw(:easy);
use Data::Dumper
Log::Log4perl->easy_init($DEBUG);

use Ropy::RopyClient;

# get a handle on the web services layer
my $client = Ropy::RopyClient->new();

my $gene_result = $client->call("genes/inorganism", "json", [ taxonID => '420245' ]);
my $gene_result_list = $gene_result->{response}->{genes};

my @genes;
# loop through the results to get the genes
# push the returned gene names onto a new array
# this array will be used as inputs to the next call
foreach my $gene_result (@{$gene_result_list}) {
    my $gene = $gene_result->{gene};
    push (@genes, genes => $gene);
}

my $result = $client->call("genes/mrnasequence", "json", [@genes]);
print Dumper ($result);
```

Calling the webservices from Javascript

The following example uses the JQuery library. It's using the JSONP callback=?, which allows the query to be executed form a Javascript file that is not served on the same domain as the server. This is good for testing purposes, but in real deployments you should serve an instance of the Crawl server on the same domain as your Javascript.

This query example does not need any other parameters:

```
$.getJSON("http://t81-omixed.internal.sanger.ac.uk:6666/organisms/list.json?callback=?", function(re
    if (result.response.organisms) {
       console.log(result.response.organisms);
    }
});
```

Here is one with parameters:

```
$.getJSON("http://t81-omixed.internal.sanger.ac.uk:6666/sourcefeatures/featureloc.json?uniqueName=Pf
   if (returned.response.features) {
       console.log(returned.response.features);
   }
});
```

More Javascript examples can be found in Tim Carver's rather excellent web-artemis project in genlib.

Calling the webservices from Python

A python client library is supplied in the ropy package (found in genlib/python) to simplify the access to the server. Here is an excerpt from the crawl test suite.

```
#!/usr/bin/env python
# encoding: utf-8
"""
post_test.py

Created by Giles Velarde on 2010-02-17.
Copyright (c) 2010 Wellcome Trust Sanger Institute. All rights reserved.
"""

import os
try:
    import simplejson as json
except ImportError:
    import json


import logging, logging.config #@UnusedImport

logger = logging.getLogger("crawl")
logging.config.fileConfig(os.path.dirname(__file__) + "/logging.conf")
```

```
from ropy.client import RopyClient

def main():
    client = RopyClient("http://localhost:6666/")

    # a POST request using the u array parameter
    parameters = {
        "u[]": ["PFA0290w:exon:2", "PFA0300c:exon:1"]
    }
    response = json.dumps(client.request("genes/featureproperties.json", parameters), indent=1)
    logger.info(response)


    # a POST request using the us splittable string parameter
    parameters2 = {
        "us": "PFA0290w:exon:2,PFA0300c:exon:1"
    }
    response2 = json.dumps(client.request("genes/featureproperties.json", parameters2), indent=1)
    logger.info(response2)


    # the GET request
    parameters3 = {
        "u": ["PFA0290w:exon:2", "PFA0300c:exon:1"]
    }
    response3 = json.dumps(client.request("genes/featureproperties.json", parameters3, False), inder
    logger.info(response3)


    assert response == response2 == response3


    # a GET request, not parsing the data, must return a string
    parameters4 = {
        "uniqueNames": ["PFA0290w:exon:2", "PFA0300c:exon:1"]
    }
    response4 = client.request("genes/featureproperties.json", parameters4, False, False)
    logger.info(response4)

    assert isinstance(response4, str)



if __name__ == '__main__':
    main()
```

On the command line

Database connection

The command line version has an optional -database parameter where you can pass in login credentials in the form of

```
-database hostname:portnumber/databasename?username
```

If you don't provide this, then a default read only connection to our database is assumed.

For obvious reasons the password should not be supplied as a command line parameter. Crawler will look for a password stored temporarily in your CRAWL_PASSWORD environment variable. This can be achieved by doing:

```
read -s -p "Enter Password: " CRAWL_PASSWORD; export CRAWL_PASSWORD
```

in the terminal before you run crawler. If you don't set a CRAWL_PASSWORD, then you will be prompted to do so by the application. However, if you're going to bsub the process that won't work, which is why crawler will only prompt for a password if your CRAWL_PASSWORD environment variable is not set.

Usage

The command line client is designed to return JSON strings, e.g.

```
crawler.py -query genes/polypeptidesequence -genes PFA0485w
```

It is designed to be fairly interactive, for instance just doing

```
$ crawler.py
```

will result in

```
Error:
Please provide a -query argument, e.g. '-query genes/list'.

Usage:  python crawler.py -query path/function [-database host:5432/database?user] [-always_return_j

Available query sets are:

        features
        genes
        organisms
        sourcefeatures
        testing
```

Drilling further with -query genes

```
$ crawler.py -query genes
Error:
Please provide 2 parts to the -query argument, the path and the function, separated by a '/', e.g. '

Usage:  python crawler.py -query path/function [-database host:5432/database?user] [-always_return_j

Available genes queries are:

        annotation_changes
        changes
        exons
        index
        inorganism
        inregion
        mrnasequence
        polypeptidesequence
        sequence
```

results in a list of queries in the query set genes that you can interrogate.

So putting in a full query path, e.g. genes/sequence

```
crawler.py -query genes/sequence
Error:
missing args: region

Usage:  python crawler.py -query path/function [-database host:5432/database?user] [-always_return_j

Available query options are:

-region the name of a region, i.e. one of the entries returned by /top.
-genes  a list of genes, for instance as supplied by the /inregion or /inorganism queries.
```

will tell you what args you need to supply. Adding the parameters:

```
crawler.py -query genes/sequence -region Pf3D7_01 -genes PFA0485w
```

should now result in a data output.

Wrapping the command line client in PERL

A Crawl module is available in genlib/perl. Here is an example of its use. As with the command-line app, you need to setup the CRAWL_PASSWORD environment variable before you launch your scripts, or else you will be prompted to do so by the application, which would be quite problematic if it's running in a bsub.

```
use strict;
use warnings;

use Crawl::Crawler;
use Crawl::Util;

my $crawler = Crawl::Crawler->new();

$crawler->database_uri("localhost:5432/pathogens?pathdb");

my $result = $crawler->query("genes/sequence", {
    region => 'Pf3D7_01',
    genes => ["PFA0485w", "PFA0570w"]
});

parse_and_print_json_data($result);
```

Example Queries

Here we list example queries. Sometimes using command line examples, sometimes using perl.

Listing organisms

On the command-line:

```
$ crawler.py -query organisms/list
{
    "response": {
        "name": "organisms/list",
        "organisms": [
            {
                "name": "Aspergillus fumigatus",
                "organism_id": "35",
                "taxonomyid": "5085"
            },
            {
                "name": "Bordetella avium 197N",
                "organism_id": "48",
                "taxonomyid": "360910"
            },
...
            {
                "name": "Yersinia pestis",
                "organism_id": "71",
                "taxonomyid": "632"
            }
        ]
    }
}
```

Using PERL:

```
use Crawl::Crawler;

my $crawler = Crawl::Crawler->new();
my $result = $crawler->query("organisms/list");

foreach my $organism (@{$result->{response}->{organisms}}) {
    print "\n" . $organism->{taxonomyid} . "\t" . $organism->{name};
}
```

Listing all the genes in an organism

On the command-line:

```
$ crawler.py -query genes/inorganism -taxonID 5833
{
    "response": {
        "genes": [
            {
                "gene": "PFB0770c",
                "type": "gene"
            },
            {
                "gene": "PFA0170c",
                "type": "gene"
            },
            {
                "gene": "PFA0315w",
                "type": "gene"
            },
            {
                "gene": "PFA0380w",
                "type": "gene"
            },
            {
                "gene": "PFA0115w",
                "type": "gene"
            },
...
            {
                "gene": "PFD0975w",
                "type": "gene"
            },
            {
                "gene": "PF14_0096",
                "type": "gene"
            }
        ],
        "name": "genes/list"
    }
}
```

Using PERL:

```
use Crawl::Crawler;

my $crawler = Crawl::Crawler->new();
my $result = $crawler->query("genes/inorganism", {taxonID => "5833"});

foreach my $gene (@{$result->{response}->{genes}}) {
    print "\n" . $gene->{gene};
}
```

Listing all the top-level regions (e.g. chromosomes, contigs) in an organism

On the command-line:

```
$ crawler.py -query regions/inorganism -taxonID 5833
{
    "response": {
        "name": "regions/inorganism",
        "regions": [
            "PFC10_API_IRAB",
            "Pf3D7_01",
            "Pf3D7_02",
            "Pf3D7_03",
            "Pf3D7_04",
            "Pf3D7_05",
            "Pf3D7_06",
            "Pf3D7_07",
            "Pf3D7_08",
            "Pf3D7_09",
            "Pf3D7_10",
            "Pf3D7_11",
            "Pf3D7_12",
            "Pf3D7_13",
            "Pf3D7_14",
            "Pf_M76611"
        ],
        "taxonId": "5833"
    }
}
```

Using PERL:

```
use Crawl::Crawler;

my $crawler = Crawl::Crawler->new();
my $result = $crawler->query("regions/inorganism", {taxonID => "5833"});

foreach my $region (@{$result->{response}->{regions}}) {
    print "\n" . $region;
}
```

Listing all the genes in a region (e.g. a contig)

On the command-line:

```
$ crawler.py -query genes/inregion -region Pf3D7_01
{
    "response": {
        "genes": [
            {
                "fmax": "313042",
                "fmin": "311947",
                "gene": "PF01TR003",
                "region": "Pf3D7_01"
            },
            {
                "fmax": "337467",
                "fmin": "336018",
                "gene": "PF01TR004",
                "region": "Pf3D7_01"
            },
            {
                "fmax": "153011",
                "fmin": "148148",
                "gene": "PFA0170c",
                "region": "Pf3D7_01"
            },
            {
                "fmax": "273641",
                "fmin": "272692",
                "gene": "PFA0315w",
                "region": "Pf3D7_01"
            },
            {
                "fmax": "320559",
                "fmin": "315773",
                "gene": "PFA0380w",
                "region": "Pf3D7_01"
            },
...
           {
                "fmax": "146630",
                "fmin": "143873",
                "gene": "PFA0165c",
                "region": "Pf3D7_01"
            }
        ],
        "name": "genes/inregion"
    }
}
```

Using PERL:

```
use Crawl::Crawler;

my $crawler = Crawl::Crawler->new();
my $result = $crawler->query("genes/inregion", {region => "Pf3D7_01"});

foreach my $gene (@{$result->{response}->{genes}}) {
    print "\n" . $gene->{gene} . "\t" . $gene->{fmin} . "\t" . $gene->{fmax};
}
```

The length of a contig

On the command-line:

```
$ crawler.py -query features/length -uniquename Pf3D7_01
{
    "response": {
        "length": 643292,
        "name": "genes/length",
        "uniquename": "Pf3D7_01"
    }
}
```

Using PERL:

```
use Crawl::Crawler;

my $crawler = Crawl::Crawler->new();
my $result = $crawler->query("features/length", {uniquename => "Pf3D7_01"});

print $result->{response}->{length};
```

The length of a gene

The request is exactly the same as that for a contig (above), but the return is more complicated because a gene may be "located" on more than one region inside the database.

On the command-line:

```
$ crawler.py -query features/length -uniquename PFA0165c
{
    "response": {
        "length": [
            {
                "feature": "PFA0165c",
                "fmax": "146630",
                "fmin": "143873",
                "length": "2757",
                "region": "Pf3D7_01"
            }
        ],
        "name": "genes/length",
        "uniquename": "PFA0165c"
    }
}
```

Using PERL:

```
use Crawl::Crawler;

my $crawler = Crawl::Crawler->new();
my $result = $crawler->query("features/length", {uniquename => "PFA0165c"});

foreach my $length (@{$result->{response}->{length}}) {
    print "\n" . $length->{feature} . "\t" . $length->{length} . "\t" . $length->{region};
}
```

Getting the sequence of all genes on a region

On the command-line:

```
$ crawler.py -query genes/sequence -region Pf3D7_01
{
    "response": {
        "name": "genes/sequence",
        "sequence": [
            {
                "feature": "PF01TR003",
                "region": "Pf3D7_01",
                "sequence": "tattatgtatatataataaataaaaaataaaatatcttttttttttcttttttttcaggattttattct
...
ttttttttttccttgcatataagatttgtttaagtatacatgatattttttcatgtatttaatgatgattttcagtttttatcatatatgttaaattaat
            }
        ]
    }
}
```

Using PERL:

```
use Crawl::Crawler;

my $crawler = Crawl::Crawler->new();
my $result = $crawler->query("genes/sequence", {region => "Pf3D7_01"});

foreach my $sequence (@{$result->{response}->{sequence}}) {
    print "\n>" . $sequence->{feature};
    print "\n" . $sequence->{sequence};
}
```

Getting the sequences of certain genes in a region

On the command-line:

```
$ crawler.py -query genes/sequence -region Pf3D7_01 -genes PFA0485w
{
    "response": {
        "name": "genes/sequence",
        "sequence": [
            {
                "feature": "PFA0485w",
                "region": "Pf3D7_01",
                "sequence": "gatgataaatattttggtaatattaataataacaattactcaagtgaaaaagaaaatttttctttgttcat
...
gttcagggtttagggttcaggatttagggttataggattcagggtttaggtttaggtttagggttcagggtttagggttcagggtttagggtttagggtt
            }
        ]
    }
}
```

Using PERL:

```
use Crawl::Crawler;

my $crawler = Crawl::Crawler->new();
my $result = $crawler->query("genes/sequence", {region => "Pf3D7_01", genes => ["PFA0485w"] });

foreach my $sequence (@{$result->{response}->{sequence}}) {
    print "\n>" . $sequence->{feature};
    print "\n" . $sequence->{sequence};
}
```

Getting the mRNA sequences of certain genes

On the command-line:

```
$ crawler.py -query genes/mrnasequence -genes PFA0485w PFA0570w
{
    "response": {
        "mrnas": [
            {
                "gene": "PFA0485w",
                "mrna": "PFA0485w.2",
                "sequence": "atgataaatattttggtaatattaataataacaattactcaagtgaaaaagaaaatttttctttgttcatg
            },
            {
                "gene": "PFA0485w",
                "mrna": "PFA0485w.1",
                "sequence": "atgataaatattttggtaatattaataataacaattactcaagtgaaaaagaaaatttttctttgttcatg
            },
            {
                "gene": "PFA0570w",
                "mrna": "PFA0570w:mRNA",
                "sequence": "atggaaggaaatgataagtgtttaaatgtcaccttggctgacatagaaaaagatacaatgaaaaataattt
            }
        ],
        "name": "genes/mrnasequence"
    }
}
```

Using PERL:

```
use Crawl::Crawler;

my $crawler = Crawl::Crawler->new();
$crawler->database_uri("localhost:5432/pathogens?pathdb");
my $result = $crawler->query("genes/mrnasequence", { genes => ["PFA0485w", "PFA0570w"] });

foreach my $mrna (@{$result->{response}->{mrnas}}) {
    print "\n>" . $mrna->{gene} . " , " . $mrna->{mrna};
    print "\n" . $mrna->{sequence};
}
```

Getting the polypeptide sequences of certain genes

On the command-line:

```
$ crawler.py -query genes/polypeptidesequence -genes PFA0485w PFA0570w
{
    "response": {
        "name": "genes/polypeptidesequence",
        "polypeptides": [
            {
                "gene": "PFA0485w",
                "polypeptide": "PFA0485w.2:pep",
                "sequence": "MINILVILIITITQVKKKIFLCSCFTYIWILLTVLCNILIVSYMYHEKNKKKTKKEKNIEKEEKKYKNKNI
            },
            {
                "gene": "PFA0485w",
                "polypeptide": "PFA0485w.1:pep",
                "sequence": "MINILVILIITITQVKKKIFLCSCFTYIWILLTVLCNILIVSYMYHEKNKKKTKKEKNIEKEEKKYKNKNI
            },
            {
                "gene": "PFA0570w",
                "polypeptide": "PFA0570w:pep",
                "sequence": "MEGNDKCLNVTLADIEKDTMKNNLSLNSKGNELLNNKKSGYKNNIKKKKKKSIKENNGNEQNKGNGTLTLK
            }
        ]
    }
}
```

Using PERL:

```
use Crawl::Crawler;

my $crawler = Crawl::Crawler->new();
my $result = $crawler->query("genes/polypeptidesequence", { genes => ["PFA0485w", "PFA0570w"] });

foreach my $polypeptide (@{$result->{response}->{polypeptides}}) {
    print "\n>" . $polypeptide->{gene} . " , " . $polypeptide->{polypeptide};
    print "\n" . $polypeptide->{sequence};
}
```

Start/stop for a feature on a region (e.g. gene on a contig)

On the command-line:

```
$ crawler.py -query features/featurecoordinates -region Pf3D7_01 -features PFA0485w PFA0570w
{
    "response": {
        "coordinates": [
            {
                "feature": "PFA0485w",
                "fmax": "389094",
                "fmin": "384437",
                "region": "Pf3D7_01"
            },
            {
                "feature": "PFA0570w",
                "fmax": "450433",
                "fmin": "447176",
                "region": "Pf3D7_01"
            }
        ],
        "name": "genes/featurecoordinates"
    }
}
```

Using PERL:

```
use Crawl::Crawler;

my $crawler = Crawl::Crawler->new();
my $result = $crawler->query("features/featurecoordinates", { region => "Pf3D7_01", features => ["PF

foreach my $coordinate (@{$result->{response}->{coordinates}}) {
    print "\n" . $coordinate->{feature} . "\t" . $coordinate->{fmin} . "\t" . $coordinate->{fmax};
}
```

Exon positions for genes on a region

On the command-line:

```
$ crawler.py -query genes/exons -region Pf3D7_01
{
    "response": {
        "coordinates": [
            {
                "exon": "PFA0485w.1:exon:14",
                "fmax": "387760",
                "fmin": "387711",
                "gene": "PFA0485w",
                "region": "Pf3D7_01"
            },
            {
                "exon": "PFA0485w.1:exon:13",
                "fmax": "387547",
                "fmin": "387484",
                "gene": "PFA0485w",
                "region": "Pf3D7_01"
            },
            {
                "exon": "PFA0485w.1:exon:12",
                "fmax": "387304",
                "fmin": "387242",
                "gene": "PFA0485w",
                "region": "Pf3D7_01"
            },
...
            {
                "exon": "PFA0355w:exon:1",
                "fmax": "298816",
                "fmin": "298769",
                "gene": "PFA0355w",
                "region": "Pf3D7_01"
            }
        ],
        "name": "genes/exons"
    }
}
```

Using PERL:

```
use Crawl::Crawler;

my $crawler = Crawl::Crawler->new();
$crawler->database_uri("localhost:5432/pathogens?pathdb");
my $result = $crawler->query("genes/exons", { region => "Pf3D7_01" });

foreach my $coordinate (@{$result->{response}->{coordinates}}) {
    print "\n" . $coordinate->{gene} . "\t" . $coordinate->{exon} . "\t" . $coordinate->{fmin} . "\t
}
```

Exon positions for selected genes on a region

On the command-line:

```
$ crawler.py -query genes/exons -region Pf3D7_01 -genes PFA0485w PFA0570w
{
    "response": {
        "coordinates": [
            {
                "exon": "PFA0485w.2:exon:18",
                "fmax": "388779",
                "fmin": "388749",
                "gene": "PFA0485w",
                "region": "Pf3D7_01"
            },
            {
                "exon": "PFA0485w.2:exon:12",
                "fmax": "387547",
                "fmin": "387484",
                "gene": "PFA0485w",
                "region": "Pf3D7_01"
            },
            {
                "exon": "PFA0485w.2:exon:11",
                "fmax": "387304",
                "fmin": "387242",
                "gene": "PFA0485w",
                "region": "Pf3D7_01"
            },
            {
                "exon": "PFA0485w.2:exon:10",
                "fmax": "387125",
                "fmin": "387058",
                "gene": "PFA0485w",
                "region": "Pf3D7_01"
            },
            {
                "exon": "PFA0485w.2:exon:9",
                "fmax": "386921",
                "fmin": "386776",
                "gene": "PFA0485w",
                "region": "Pf3D7_01"
            },
            {
                "exon": "PFA0485w.2:exon:8",
                "fmax": "386674",
                "fmin": "386614",
                "gene": "PFA0485w",
                "region": "Pf3D7_01"
            },
            {
                "exon": "PFA0485w.2:exon:7",
                "fmax": "386539",
                "fmin": "386417",
                "gene": "PFA0485w",
                "region": "Pf3D7_01"
            },
            {
                "exon": "PFA0485w.2:exon:6",
                "fmax": "386253",
                "fmin": "386050",
                "gene": "PFA0485w",
                "region": "Pf3D7_01"
            },
            {
                "exon": "PFA0485w.2:exon:5",
                "fmax": "385961",
                "fmin": "385780",
                "gene": "PFA0485w",
                "region": "Pf3D7_01"
            },
            {
                "exon": "PFA0485w.2:exon:4",
                "fmax": "385692",
                "fmin": "385553",
                "gene": "PFA0485w",
                "region": "Pf3D7_01"
            },
            {
                "exon": "PFA0485w.2:exon:3",
                "fmax": "385424",
                "fmin": "385315",
                "gene": "PFA0485w",
                "region": "Pf3D7_01"
            },
            {
                "exon": "PFA0485w.2:exon:2",
                "fmax": "385166",
                "fmin": "385040",
                "gene": "PFA0485w",
                "region": "Pf3D7_01"
            },
            {
                "exon": "PFA0485w.2:exon:1",
                "fmax": "384876",
                "fmin": "384437",
                "gene": "PFA0485w",
                "region": "Pf3D7_01"
            },
            {
                "exon": "PFA0485w.2:exon:16",
                "fmax": "388453",
                "fmin": "388367",
                "gene": "PFA0485w",
                "region": "Pf3D7_01"
            },
            {
                "exon": "PFA0485w.2:exon:15",
                    "fmax": "388196",
                    "fmin": "388024",
                    "gene": "PFA0485w",
                    "region": "Pf3D7_01"
                },
                {
                    "exon": "PFA0485w.2:exon:14",
                    "fmax": "387876",
                    "fmin": "387844",
                    "gene": "PFA0485w",
                    "region": "Pf3D7_01"
                },
                {
                    "exon": "PFA0485w.2:exon:13",
                    "fmax": "387760",
                    "fmin": "387711",
                    "gene": "PFA0485w",
                    "region": "Pf3D7_01"
                },
                {
                    "exon": "PFA0485w.1:exon:20",
                    "fmax": "389094",
                    "fmin": "389027",
                    "gene": "PFA0485w",
                    "region": "Pf3D7_01"
                },
                {
                    "exon": "PFA0485w.1:exon:19",
                    "fmax": "388885",
                    "fmin": "388749",
                    "gene": "PFA0485w",
                    "region": "Pf3D7_01"
                },
                {
                    "exon": "PFA0485w.1:exon:18",
                    "fmax": "388616",
                    "fmin": "388588",
                    "gene": "PFA0485w",
                    "region": "Pf3D7_01"
                },
                {
                    "exon": "PFA0485w.1:exon:17",
                    "fmax": "388453",
                    "fmin": "388367",
                    "gene": "PFA0485w",
                    "region": "Pf3D7_01"
                },
                {
                    "exon": "PFA0485w.1:exon:16",
                    "fmax": "388196",
                    "fmin": "388024",
                    "gene": "PFA0485w",
                    "region": "Pf3D7_01"
                },
                {
                    "exon": "PFA0485w.1:exon:15",
                    "fmax": "387876",
                    "fmin": "387844",
                    "gene": "PFA0485w",
                    "region": "Pf3D7_01"
                },
                {
                    "exon": "PFA0485w.1:exon:14",
                    "fmax": "387760",
                    "fmin": "387711",
                    "gene": "PFA0485w",
                    "region": "Pf3D7_01"
                },
                {
                    "exon": "PFA0485w.1:exon:13",
                "fmax": "387547",
                "fmin": "387484",
                "gene": "PFA0485w",
                "region": "Pf3D7_01"
            },
            {
                "exon": "PFA0485w.1:exon:12",
                "fmax": "387304",
                "fmin": "387242",
                "gene": "PFA0485w",
                "region": "Pf3D7_01"
            },
            {
                "exon": "PFA0485w.1:exon:11",
                "fmax": "387125",
                "fmin": "387058",
                "gene": "PFA0485w",
                "region": "Pf3D7_01"
            },
            {
                "exon": "PFA0485w.1:exon:10",
                "fmax": "386921",
                "fmin": "386776",
                "gene": "PFA0485w",
                "region": "Pf3D7_01"
            },
            {
                "exon": "PFA0485w.1:exon:9",
                "fmax": "386674",
                "fmin": "386614",
                "gene": "PFA0485w",
                "region": "Pf3D7_01"
            },
            {
                "exon": "PFA0485w.1:exon:8",
                "fmax": "386539",
                "fmin": "386417",
                "gene": "PFA0485w",
                "region": "Pf3D7_01"
            },
            {
                "exon": "PFA0485w.1:exon:7",
                "fmax": "386253",
                "fmin": "386050",
                "gene": "PFA0485w",
                "region": "Pf3D7_01"
            },
            {
                "exon": "PFA0485w.1:exon:6",
                "fmax": "385961",
                "fmin": "385780",
                "gene": "PFA0485w",
                "region": "Pf3D7_01"
            },
            {
                "exon": "PFA0485w.1:exon:5",
                "fmax": "385692",
                "fmin": "385553",
                "gene": "PFA0485w",
                "region": "Pf3D7_01"
            },
            {
                "exon": "PFA0485w.1:exon:4",
                "fmax": "385424",
                "fmin": "385315",
                "gene": "PFA0485w",
                "region": "Pf3D7_01"
            },
            {
                "exon": "PFA0485w.1:exon:3",
            "fmax": "385166",
            "fmin": "385040",
            "gene": "PFA0485w",
            "region": "Pf3D7_01"
        },
        {
            "exon": "PFA0485w.1:exon:2",
            "fmax": "384876",
            "fmin": "384437",
            "gene": "PFA0485w",
            "region": "Pf3D7_01"
        },
        {
            "exon": "PFA0570w:exon:1",
            "fmax": "447926",
            "fmin": "447176",
            "gene": "PFA0570w",
            "region": "Pf3D7_01"
        },
        {
            "exon": "PFA0570w:exon:2",
            "fmax": "448167",
            "fmin": "448042",
            "gene": "PFA0570w",
            "region": "Pf3D7_01"
        },
        {
            "exon": "PFA0570w:exon:9",
            "fmax": "448356",
            "fmin": "448268",
            "gene": "PFA0570w",
            "region": "Pf3D7_01"
        },
        {
            "exon": "PFA0570w:exon:3",
            "fmax": "448573",
            "fmin": "448489",
            "gene": "PFA0570w",
            "region": "Pf3D7_01"
        },
        {
            "exon": "PFA0570w:exon:4",
            "fmax": "448898",
            "fmin": "448767",
            "gene": "PFA0570w",
            "region": "Pf3D7_01"
        },
        {
            "exon": "PFA0570w:exon:5",
            "fmax": "449013",
            "fmin": "448991",
            "gene": "PFA0570w",
            "region": "Pf3D7_01"
        },
        {
            "exon": "PFA0570w:exon:6",
            "fmax": "449340",
            "fmin": "449177",
            "gene": "PFA0570w",
            "region": "Pf3D7_01"
        },
        {
            "exon": "PFA0570w:exon:7",
            "fmax": "449970",
            "fmin": "449541",
            "gene": "PFA0570w",
            "region": "Pf3D7_01"
        },
        {
            "exon": "PFA0570w:exon:8",
                "fmax": "450433",
                "fmin": "450251",
                "gene": "PFA0570w",
                "region": "Pf3D7_01"
            }
        ],
        "name": "genes/exons"
    }
}
```

Using PERL:

```
use Crawl::Crawler;

my $crawler = Crawl::Crawler->new();
$crawler->database_uri("localhost:5432/pathogens?pathdb");
my $result = $crawler->query("genes/exons", { region => "Pf3D7_01", genes => ["PFA0485w", "PFA0570w"

foreach my $coordinate (@{$result->{response}->{coordinates}}) {
    print "\n" . $coordinate->{gene} . "\t" . $coordinate->{exon} . "\t" . $coordinate->{fmin} . "\t
}
```

GO terms for a gene

On the command-line:

```
$ crawler.py -query features/terms -features PFA0570w:pep -controlled_vocabularies biological_proces
{
    "response": {
        "features": [
            {
                "feature": "PFA0570w:pep",
                "terms": [
                    {
                        "accession": "0005737",
                        "cv": "cellular_component",
                        "cvterm": "cytoplasm",
                        "dbxrefs": [],
                        "is_not": "False",
                        "props": [
                            {
                                "prop": "20100131",
                                "proptype": "date",
                                "proptypecv": "feature_property"
                            },
                            {
                                "prop": "Inferred from Electronic Annotation",
                                "proptype": "evidence",
                                "proptypecv": "genedb_misc"
                            },
                            {
                                "prop": "From iprscan",
                                "proptype": "autocomment",
                                "proptypecv": "genedb_misc"
                            }
                        ],
                        "pubs": []
                    },
                    {
                        "accession": "0006417",
                        "cv": "biological_process",
                        "cvterm": "regulation of translation",
                        "dbxrefs": [],
                    "is_not": "False",
                    "props": [
                        {
                            "prop": "From GO association file",
                            "proptype": "autocomment",
                            "proptypecv": "genedb_misc"
                        },
                        {
                            "prop": "GeneDB_Pfalciparum",
                            "proptype": "attribution",
                            "proptypecv": "genedb_misc"
                        },
                        {
                            "prop": "Inferred from Electronic Annotation",
                            "proptype": "evidence",
                            "proptypecv": "genedb_misc"
                        },
                        {
                            "prop": "20090805",
                            "proptype": "date",
                            "proptypecv": "feature_property"
                        }
                    ],
                    "pubs": [
                        {
                            "accession": "19435743",
                            "database": "PMID"
                        }
                    ]
                },
                {
                    "accession": "0006413",
                    "cv": "biological_process",
                    "cvterm": "translational initiation",
                    "dbxrefs": [],
                    "is_not": "False",
                    "props": [
                        {
                            "prop": "From GO association file",
                            "proptype": "autocomment",
                            "proptypecv": "genedb_misc"
                        },
                        {
                            "prop": "GeneDB_Pfalciparum",
                            "proptype": "attribution",
                            "proptypecv": "genedb_misc"
                        },
                        {
                            "prop": "Inferred from Electronic Annotation",
                            "proptype": "evidence",
                            "proptypecv": "genedb_misc"
                        },
                        {
                            "prop": "20090805",
                            "proptype": "date",
                            "proptypecv": "feature_property"
                        }
                    ],
                    "pubs": [
                        {
                            "accession": "19435743",
                            "database": "PMID"
                        }
                    ]
                },
                {
                    "accession": "0006412",
                    "cv": "biological_process",
                    "cvterm": "translation",
                    "dbxrefs": [],
                            "is_not": "False",
                            "props": [
                                {
                                    "prop": "From GO association file",
                                    "proptype": "autocomment",
                                    "proptypecv": "genedb_misc"
                                },
                                {
                                    "prop": "GeneDB_Pfalciparum",
                                    "proptype": "attribution",
                                    "proptypecv": "genedb_misc"
                                },
                                {
                                    "prop": "Inferred from Electronic Annotation",
                                    "proptype": "evidence",
                                    "proptypecv": "genedb_misc"
                                },
                                {
                                    "prop": "20090805",
                                    "proptype": "date",
                                    "proptypecv": "feature_property"
                                }
                            ],
                            "pubs": [
                                {
                                    "accession": "19435743",
                                    "database": "PMID"
                                }
                            ]
                        },
                        {
                            "accession": "0003743",
                            "cv": "molecular_function",
                            "cvterm": "translation initiation factor activity",
                            "dbxrefs": [],
                            "is_not": "False",
                            "props": [
                                {
                                    "prop": "From GO association file",
                                    "proptype": "autocomment",
                                    "proptypecv": "genedb_misc"
                                },
                                {
                                    "prop": "GeneDB_Pfalciparum",
                                    "proptype": "attribution",
                                    "proptypecv": "genedb_misc"
                                },
                                {
                                    "prop": "Inferred from Electronic Annotation",
                                    "proptype": "evidence",
                                    "proptypecv": "genedb_misc"
                                },
                                {
                                    "prop": "20090805",
                                    "proptype": "date",
                                    "proptypecv": "feature_property"
                                }
                            ],
                            "pubs": [
                                {
                                    "accession": "19435743",
                                    "database": "PMID"
                                }
                            ]
                        },
                        {
                            "accession": "0003723",
                            "cv": "molecular_function",
                            "cvterm": "RNA binding",
                            "dbxrefs": [],
                        "is_not": "False",
                        "props": [
                            {
                                "prop": "From GO association file",
                                "proptype": "autocomment",
                                "proptypecv": "genedb_misc"
                            },
                            {
                                "prop": "GeneDB_Pfalciparum",
                                "proptype": "attribution",
                                "proptypecv": "genedb_misc"
                            },
                            {
                                "prop": "Inferred from Electronic Annotation",
                                "proptype": "evidence",
                                "proptypecv": "genedb_misc"
                            },
                            {
                                "prop": "20090805",
                                "proptype": "date",
                                "proptypecv": "feature_property"
                            }
                        ],
                        "pubs": [
                            {
                                "accession": "19435743",
                                "database": "PMID"
                            }
                        ]
                    }
                ]
            }
        ],
        "name": "genes/featurecvterms"
    }
}
```

Using PERL:

```
use Crawl::Crawler;

my $crawler = Crawl::Crawler->new();
my $result = $crawler->query("features/terms", {
    features => ["PFA0570w:pep"],
    controlled_vocabularies => ["biological_process", "molecular_function", "cellular_component"]
});

foreach my $feature (@{$result->{response}->{features}}) {
    print "\n" . $feature->{feature};
    foreach my $term (@{$feature->{terms}}) {
        print "\n" . $term->{cv} . "\t" . $term->{cvterm} . "\t" . $term->{accession} . "\t" . $term
    }
}
```

Getting orthologues of a gene

On the command-line:

```
$ crawler.py -query features/orthologues -features PKH_041570:pep
```

```
    "response": {
        "features": [
            {
                "feature": "PKH_041570:pep",
                "orthologues": [
                    {
                        "feature": "PKH_041570:pep",
                        "genus": "Plasmodium",
                        "ortho": "PKH_081810:pep",
                        "species": "knowlesi",
                        "value": "5850"
                    },
                    {
                        "feature": "PKH_041570:pep",
                        "genus": "Plasmodium",
                        "ortho": "PKH_082520:pep",
                        "species": "knowlesi",
                        "value": "5850"
                    },
...
                    {
                        "feature": "PKH_041570:pep",
                        "genus": "Plasmodium",
                        "ortho": "PKH_146740:pep",
                        "species": "knowlesi",
                        "value": "5850"
                    }
                ]
            }
        ],
        "name": "genes/orthologues"
    }
}
```

Using PERL:

```
use Crawl::Crawler;

my $crawler = Crawl::Crawler->new();
$crawler->database_uri("localhost:5432/pathogens?pathdb");
my $result = $crawler->query("features/orthologues", {
    features => ["PKH_041570:pep"]
});

foreach my $feature (@{$result->{response}->{features}}) {
    print "\n" . $feature->{feature};
    foreach my $term (@{$feature->{orthologues}}) {
        print "\n" . $term->{feature} . "\t" . $term->{ortho} . "\t" . $term->{genus} . "\t" . $term
    }
}
```

Architecture

The crawl system is a dual command-line web-application. It runs ontop of the Ropy library (also available in genlib), which itself depends on a Python stack made up of the CherryPy (http://www.cherrypy.org/) web server and the Psycopg2 (http://initd.org/psycopg/) Postgres adapter, with some templating done using Cheetah (http://www.cheetahtemplate.org/) .

The system has been designed to be very simple. At the time of writing ropy.server only has about 5 classes, and ropy.query the same. The crawl.api.controllers in crawl all extend ropy's RESTController, and currently there are only 5 or 6 of those, but it is expected that number will grow as new kinds of query are required.

Underneath the controllers there is a single crawl.api.db.Query class, that handles all the SQL calls. Each SQL call is specified in a seperate SQL file, and all SQL files can be found in the crawl sql folders.

That's pretty much it. There's no indexing, caching, and no Object Relational Mapping (ORM). It would be possible to add an ORM quite easily. And some experiments have been done with Django's ORM and SQLAlchemy. At this stage there doesn't seem to be a need for an ORM because most of the queries crawl is running benefit tremendously from being able to hand-craft the SQL.

Installation on a Mac

Installing the Python libraries

Crawl and Ropy libraries are installed on our systems, and a crawl server is deployed on t81-omixed. However, you may want to try it out on your Mac, and so here are some instructions. These don't cover installing Chado.

```
# update setuptools to the latest version
sudo easy_install -U setuptools

# install dependencies
sudo easy_install cherrypy
sudo easy_install cheetah
sudo easy_install simplejson
sudo easy_install routes


# this may need to be set to 8.4 if that's what you have installed
PATH=$PATH:/Library/PostgreSQL/8.3/bin/
sudo easy_install psycopg2

# you'll probably need to add this to your .profile in the future, but you can run it on the commanc
export PYTHONPATH=/path/to/your/python/script/folder/:$PYTHONPATH

$ cd /path/to/your/python/script/folder/

# checkout server and the microframework
$ svn co svn+ssh://svn.internal.sanger.ac.uk/repos/svn/pathsoft/psu/trunk/genlib/python/ropy
$ svn co svn+ssh://svn.internal.sanger.ac.uk/repos/svn/pathsoft/psu/trunk/genlib/python/crawl
$ svn co svn+ssh://svn.internal.sanger.ac.uk/repos/svn/pathsoft/psu/trunk/genlib/python/util

$ cd crawl

# rename the appconf
$ cp ini/appconf.py.default ini/appconf.py
$ vim ini/appconf.py

# edit the logging configuration file, there's a line at the bottom where you point to where to log
$ vim ini/logging.py

$ python server.py -a ini/appconf.py -s ini/serverconf.py -l ini/logging.py
```

Installing the PERL wrapper libraries

For this you will need to checkout the PERL genlib folder,

```
$ svn co svn+ssh://svn.internal.sanger.ac.uk/repos/svn/pathsoft/psu/trunk/genlib/perl/src
```

and add this path to your PERL5LIB, e.g. :

```
$ export PERL5LIB=/Volumes/us/db/gv1/code/genlib/perl/src:$PERL5LIB
```

Its dependencies are all available via CPAN:

- Log4perl
- JSON
- JSON::XS

Warning: The Python password prompt does not work on OSX when wrapped by PERL

We have noticed, on OSX, that when calling the PERL modules, if an appropriate CRAWL_PASSWORD is not set beforehand, that the Python crawler.py will prompt for a password, but this won't appear in the shell, as it normally would do in Linux. For that reason, be sure to set your CRAWL_PASSWORD before running any PERL scripts that use Crawl::Crawler.

See the database connection section for details of how to set this up.

Retrieved from "http://mediawiki.internal.sanger.ac.uk/wiki/index.php/Crawl"

- This page was last modified on 7 April 2010, at 13:00.